Joel Devey
A02094591

Hackathon Reflection

I participated in USU's 2019 hackathon this year. I was on a team with Steven Scott. We both stayed for about five hours the first night, then came back the next day and worked for about three hours. We also worked on the project a bit later that day. Since we both had other commitments that day, we weren't able to submit our project for judging.

Together, we did a data visualization in which we used physics to display a graph in a maximally appealing fashion. It is worth noting that the core idea of the project was not ours. We originally heard it from Dr. Edwards, a computer science professor at USU who specializes in data visualization. However, we were happy to implement it in our own way and add a few extra features. It was not used for any assignment or project in the data visualization class.

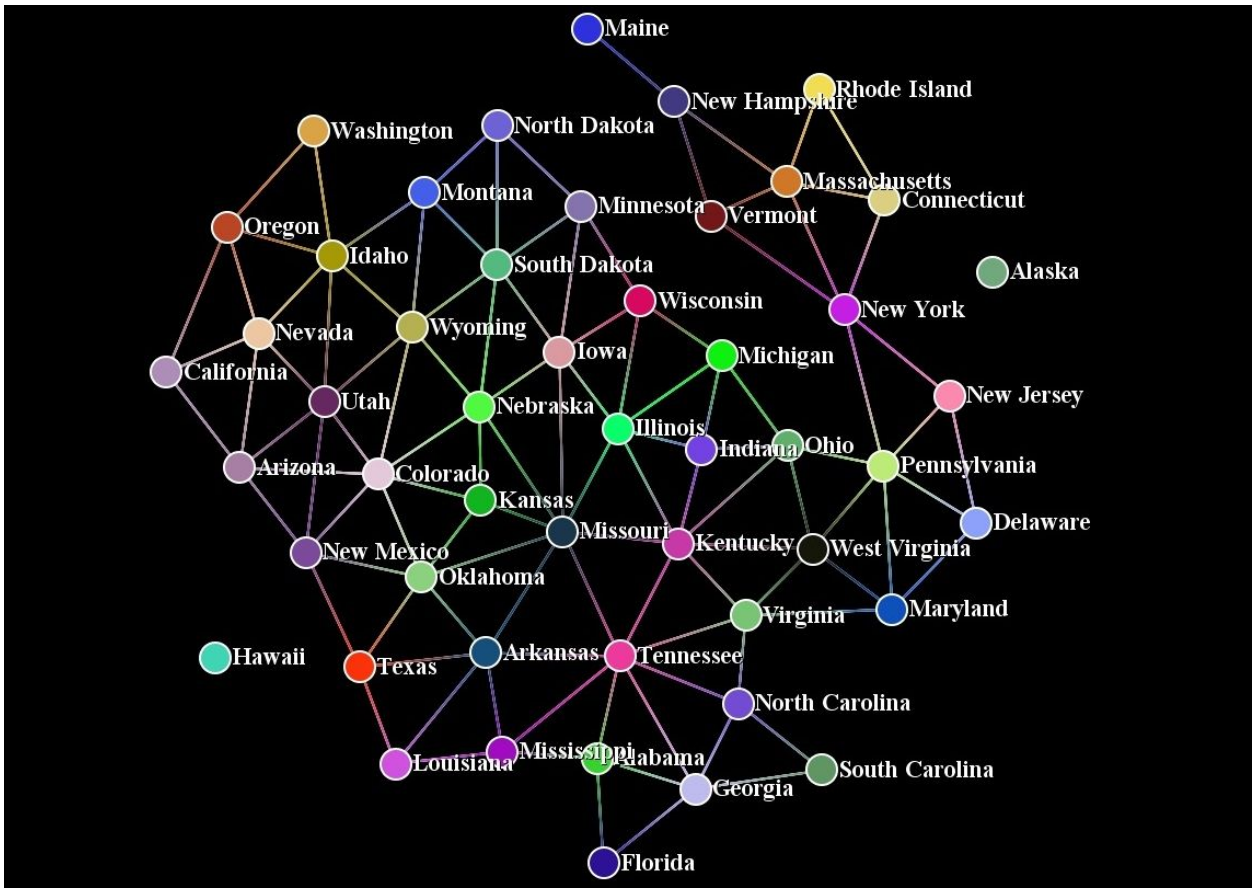
To summarize what this project aims to accomplish, it is worth diving into an important problem in graph theory and data visualization. Given a directed or undirected graph (network) that consists of nodes and edges, how can we display it in such a way that is visually appealing (i.e. has as few intersections as possible)? Solving this problem perfectly (finding the intersection number of a graph, or the minimum number of edge intersections in a two-dimensional layout of a graph) is NP-hard, so it makes sense to try an approximation.

How does our approximation work? We apply two main forces: springs and magnets. We visualize each edge as a spring, while we visualize each node as a magnet, where each magnet always repels other magnets. Since the force created by a spring is linear with respect to the distance it is stretched (Hooke's law), and the force between two magnets decreases squarely with respect to the distance between them, nodes will naturally settle upon a certain equilibrium dependent upon the constants supplied in the physics simulation. If two nodes with an edge between them grow too far apart, the spring force will pull them together. If they grow too close, they will be repelled by magnetism.

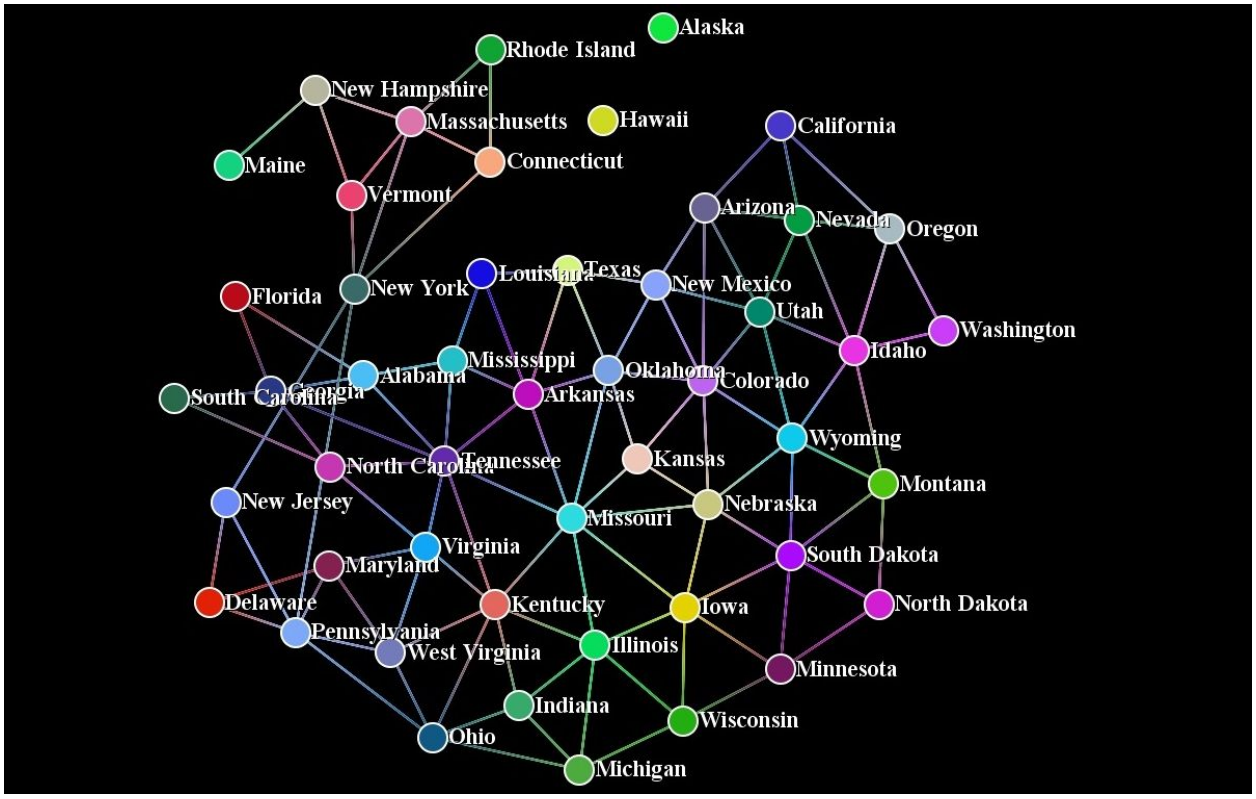
By randomly placing the nodes in a two-dimensional space with their edges intact, including the concept of velocity (rather than a static movement each tick), and a slight gravity well to pull nodes towards the center of the screen, we were able to create a simulation that

produced good approximate solutions for various networks. While we did not try a huge variety of datasets, we were able to try a few. Our three favorite datasets were 1) US states as nodes, with borders shared as edges, 2) Les Misérables characters as nodes, with character interactions as edges, and 3) members of a university karate club as nodes, with friendships as edges. You can view these data sources (gml files) in the data directory at the root of our project.

A particularly successful run for the US states dataset is shown below:



In this run, there is only one intersection, and it's between Utah / New Mexico and Arizona / Colorado. This can't be fixed since our dataset considered each of the "four corner" states as connected to one another. Hence, this run produced a result that is as good as possible given the dataset. A less successful run is shown below:



Though not bad, this result has more intersections than the previous run. These simulations were run with identical physics, but the differences in starting positions produced different results.

Datasets with more edges and nodes proved far more difficult to display in an appealing fashion. The Les Misérables dataset was more complex than the US states dataset, and while it was still able to convey useful information, I would consider it to be on the large end of what I would consider to be acceptable without a larger area for displaying the data. It never came close to resolving, and I strongly suspect that it didn't have any arrangements with no intersections; some characters (such as Jean Valjean) simply had too many connections. In my project root folder, you may look in the "demos" directory for videos of both the US states dataset and the Les Misérables dataset in action.

I would estimate that our contributions to the project were approximately equal, and we both worked on all aspects of the project, though I was slightly more focused on the display and user interface than he was. The physics engine was built from scratch, and I believe it was made easier by nature of having both of us working on it, rather than just one of us.

We chose to build our project in Java using Swing and Awt. While we had a bit of a learning curve, they proved to do the job well, and we didn't feel as though we were being hindered by our choice of framework.

As we built the project, I applied lessons about AME principles. We did our best to ensure that each class had functions that enacted changes on their own member variables, rather than the member variables of other classes (encapsulation). We divided the state of the graph into distinct datagrams. For example, we made a class for nodes that contained only the node's position, velocity, id, label, and nothing else, which I believe was a good way to have good localization of design decisions. We made a file for storing all of our display and physics constants to avoid repeating ourselves (DRY principle).

We didn't see a lot of places to apply software engineering patterns, other than the observer pattern, which we did implement. We decided that the state of the graph was a key piece of information that several other parts of the program may want to "observe". As such, we made two observers: a visual display observer to render the state of the graph to the screen, and a file output observer that would periodically write the state of the graph to a file for debugging or mathematical analysis. This worked smoothly, and I was glad to have learned about the observer pattern, since I believe it is non-trivial enough that I wouldn't have thought to do as such on my own.

I learned many things from the hackathon, and found it to be very enjoyable. I learned a lot about physics simulations, and how to make them stable and predictable. I also learned a lot about how to apply basic object-oriented programming principles, even when there aren't any complex patterns that appear to be in need of use. I also learned how to create a small / medium-sized project in a very short period of time in a small group. Even working in a group of two is a very different experience from working alone. We still had to work out plenty of merge conflicts! Overall, it was one of my favorite projects I've ever worked on, and I'm happy with the results.